592698

Two solvers for tractable temporal constraints with preferences

F. Rossi[1], K.B. Venable[1], L. Khatib[2,3], P. Morris[3], R. Morris[3]

[1] Department of Pure and Applied Mathematics, University of Padova, Italy. E-mail: frossi@math.unipd.it, kvenable@math.unipd.it

[2] Kestrel Technology

[3] NASA Ames Research Center, Moffett Field, CA, USA. E-mail: {lina,pmorris,morris}@ptolemy.arc.nasa.gov

Abstract

A number of reasoning problems involving the manipulation of temporal information can naturally be viewed as implicitly inducing an ordering of potential local decisions involving time on the basis of preferences. Soft temporal constraints problems allow to describe in a natural way scenarios where events happen over time and preferences are associated to event distances and durations.

In general, solving soft temporal problems require exponential time in the worst case, but there are interesting subclasses of problems which are polynomially solvable. We describe two solvers based on two different approaches for solving the same tractable subclass. For each solver we present the theoretical results it stands on, a description of the algorithm and some experimental results. The random generator used to build the problems on which tests are performed is also described. Finally, we compare the two solvers highlighting the tradeoff between performance and representational power.

Introduction and motivation

Several real world problems involving the manipulation of temporal information in order to find an assignment of times to a set of activities or events can naturally be viewed as having preferences associated with local temporal decisions, where by a local temporal decision we mean one associated with how long a single activity should last, when it should occur, or how it should be ordered with respect to other activities.

For example, an antenna on an earth orbiting satellite such as Landsat 7 must be slewed so that it is pointing at a ground station in order for recorded science or telemetry data to be downlinked to earth. Antenna slewing on Landsat 7 has been shown to occasionally cause a slight vibration to the satellite, which in turn might affect the quality of the image taken by the scanning instrument if the scanner is in use during slewing. Consequently, it is preferable for the slewing activity not to overlap any scanning activity, although because the detrimental effect on image quality occurs only intermittently, this disjointness is best not expressed as a hard constraint. This is only one of the many real world problems that can be casted and, under certain assumptions, solved in our framework .

This paper presents the current formalism and results for soft temporal constraint problems, and describes two machineries we have developed for solving such problems. The implemented modules rely on the theoretical results (such as those on tractability of some classes of problems) and make some assumptions for both tractability and efficiency. In particular: (1) both solvers are able to deal with soft temporal constraints with one interval per constraint, and with a particular shape of the preference functions, which assures tractability (like for Simple Temporal Constraints in the case of hard constraints (?)); (2) preferences are dealt with via a fuzzy (max-min) framework; (3) our random problem generator is based on some parameters to generate a soft temporal problem, which suitably extend the usual ones for hard CSPs (density, tightness, ...).

Some preliminary ideas which have led to the development described in this paper have been presented in (?).

Temporal constraint problems with preferences

Temporal constraint reasoning. In the Temporal CSP framework (TCSP) (?), variables represent events happening over time, and each constraint gives an allowed range for the distances or durations, expressed as a set of intervals over the time line. Satisfying such a constraint means choosing any of the allowed distances. A solution for a TCSP consisting of a set of temporal constraints is an assignment of values to its variables such that all constraints are satisfied.

As expected, general TCSPs are NP-hard. However, TCSPs with just one interval for each constraint, called STPs, are polynomially solvable (see (?) for details).

Although very expressive, TCSPs are able to model just *hard* temporal constraints. This means that all constraints have to be satisfied, and that the solutions of a constraint are all equally satisfying. However, in many real-life some solutions are preferred with respect to others. Therefore the global problem is not to find a way to satisfy all constraints, but to find a way to satisfy them optimally, according to the preferences specified.

To address such problems, recently (?) a new framework has been proposed, where each temporal constraint is associated with a preference function, which specifies the preference for each distance. This framework is based on a simple

merge of TCSPs and soft constraints, where for soft constraints we have taken a general framework based on semirings (?). The result is a class of problems called Temporal Constraint Satisfaction problems with preferences (TCSPPs).

A *soft temporal constraint* in a TCSPP is represented by a pair consisting of a set of disjoint intervals and a preference function: $\langle I = \{[a_1, b_1], \ldots, [a_n, b_n]\}, f\rangle$, where $f : I^1 \rightarrow A$, is a mapping of the elements of I into preference values, taken from a set A.

A *solution* to a TCSPP is a complete assignment to all the variables that satisfies the distance constraints. Each solution has a *global preference value*, obtained by combining the local preference values found in the constraints. To formalize the process of combining local preferences into a global preference, and comparing solutions, we impose a semiring structure ont the TCSPP framework.

A *semiring* is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that A is a set and $\mathbf{0}, \mathbf{1} \in A$; $+$, the additive operation, is commutative, associative and $\mathbf{0}$ is its unit element; $\times$, the multiplicative operation, is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element. A *c-semiring* is a semiring in which $+$ is idempotent, $\mathbf{1}$ is its absorbing element, and $\times$ is commutative. These additional properties (w.r.t. usual semirings) are required to cope with the usual nature of constraints.

C-semirings allow for a partial order relation $\leq_S$ over A to be defined as $a \leq_S b$ iff $a+b = b$. Informally, $\leq_S$ gives us a way to compare tuples of values and constraints, and $a \leq_S b$ can be read *b is better than a*. Moreover, one can prove that for all $a, b \in A$, $a + b$ is the least upper bound (lub) of a and b; and if $\times$ is idempotent, then $\langle A, \leq_S\rangle$ is a complete distributive lattice and $\times$ is its greatest lower bound (*glb*).

Given a semiring[2] with a set of values A, each preference function f associated with a soft constraint $\langle I, f\rangle$ of a TCSPP takes an element from I and returns an element of A, where A is the carrier of a semiring. This allows us to associate a preference with a duration or a distance. The two semiring operations allow for complete solutions to be evaluated in terms of the preference values assigned locally. More precisely, given a solution t in a TCSPP with associated semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, let $T_{ij} = \langle I_{i,j}, f_{i,j}\rangle$ be a soft constraint over variables X_i, X_j and (v_i, v_j) be the projection of t over the values assigned to variables X_i and X_j (abbreviated as $(v_i, v_j) = t_{\downarrow X_i, X_j}$). Then, the corresponding preference value given by f_{ij} is $f_{ij}(v_j - v_i)$, where $v_j - v_i \in I_{i,j}$. Finally, where $F = \{x_1, \ldots, x_k\}$ is a set, and $\times$ is the multiplicative operator on the semiring, let $\times F$ abbreviate $x_1 \times \ldots \times x_k$. Then the global preference value of t, $val(t)$, is defined to be $val(t) = \times\{f_{ij}(v_j - v_i) \mid (v_i, v_j) = t_{\downarrow X_i, X_j}\}$. The optimal solutions of a TCSPP are those solutions which have the best global preference value, where "best" is determined by the ordering $\leq_S$ of the values in the semiring.

[1]Here by I we mean the set of all elements appearing in the intervals of I.

[2]For simplicity, from now on we will write *semiring* meaning *c-semiring*.

The semiring underlying the problems targeted here is $S_{fuzzy} = \langle [0, 1], max, min, 0, 1\rangle$, used for fuzzy constraint solving (?). The global preference value of a solution will be the minimum of all the preference values associated with the distances selected by this solution in all constraints, and the best solutions will be those with the maximal value.

A special case occurs when each constraint of a TCSPP contains a single interval. We call such problems *Simple Temporal Problems with Preferences* (STPPs). In (?) it has been shown that, while in general TCSPPs are NP-hard, under certain restrictions on the "shape" of the preference functions and on the semiring, STPPs are tractable.

A *semi-convex* function f is one such that, for all Y, the set $\{X$ such that $f(X) \geq Y\}$ forms an interval. It is easy to see that semi-convex functions include linear ones, as well as convex and some step functions. For example, the *close to k* criteria cannot be coded into a linear preference function, but it can be easily specified by a semi-convex preference function.

It is proven in (?) that STPPs with semi-convex preference functions and a semiring with a total order of preference values and an idempotent multiplicative operation can be solved in polynomial time.

A solving module based on path consistency

The tractability results for STPPs can be translated in practice as follows: to find an optimal solution for an STPP, we can first apply path-consistency (suitably adapted to STPPs, see (?)) and then use a search procedure to find a solution without the need to backtrack. More in details, it is possible to show that: (1) Semi-convex functions are closed w.r.t. path-consistency: if we start from an STPP P with semi-convex functions, and we apply path-consistency, we get a new STPP P' with semi-convex functions (see (?)). The only difference in the two problems is that the new one can have smaller intervals and worse preference values in the preference functions.(2)After applying path-consistency, all preference functions in P' have the same best preference level.(3) Consider the STP obtained from the STPP P' by taking, for each constraint, the sub-interval corresponding to the best preference level; then, the solutions of such an STP coincide with the best solutions of the original P (and also of P'). Therefore, finding a solution of this STP means finding an optimal solution of P. Our first solving module, which we call path-solver, relies on these results. In fact, the STPP solver takes as input an STPP with semi-convex preference functions, and returns an optimal solution of the given problem, working as follows and as shown in Figure ??: first, path-consistency is applied to the given problem, by function STPP_PC-2, producing a new problem P'; then, an STP corresponding to P' is constructed, applying REDUCE_TO_BEST to P', by taking the subintervals corresponding to the best preference level and forgetting about the preference functions; finally, a backtrack-free search is performed to find a solution of the STP, specifically the earliest one is returned by function EARLIEST_BEST. All these steps are polynomial, so the overall complexity of solving an STPP with the above assumptions is polynomial.

Algoritmo path-solver

1. **input** STPP P ;
2. STPP P'=STPP_PC-2(P);
3. **if** P' inconsistent **then** exit;
4. STP P"=REDUCE_TO_BEST(P');
5. **return** EARLIEST_BEST(P").

Figure 1: Pseudocode for path-solver.

This STPP solver has been tested both on toy problems and on randomly-generated problems. The random generator we have developed focusses on a particular subclass of semi-convex preference functions: convex quadratic functions of the form $ax^2 + bx + c$, with $a \leq 0$. The choice has been suggested both by the expressiveness of such a class of functions and also by the facility of expressing functions in this class (just three parameters). Moreover, it generates fuzzy STPPs, thus preference values are between 0 and 1.

An STPP is generated according to the value of the following parameters:

- number n of variables;
- range r for the initial solution: to assure that the generated problem has at least one solution, we first generate such a solution, by giving to each variable a random value within the range $[0, r]$;
- density: percentage of constraints that are not universal (that is, with the maximum range and preference 1 for all interval values);
- maximum expansion from initial solution (max): for each constraint, the bounds of its interval are set by using a random value between 0 and max, to be added to and subtracted from the timepoint identified for this constraint by the initial solution;
- perturbation of preference functions (pa, pb, pc): we recall that each preference function can be described by three values (a, b, and c); to set such values for each constraint, the generator starts from a standard quadratic function which passes through the end points of the interval, with value 0, and the middlepoint, with value 0.5, and then modifies it according to the percentages specified for a, b, and c.

For example, if we call the generator with the parameters $\langle 10, 20, 30, 40, 20, 25, 30\rangle$, it will generate a fuzzy STPP with 10 variables. Moreover, the initial solution will be chosen by giving to each variable a value between 0 and 20. Among all the constraints, 70% of them will be universal, while the other 30% will be specified as follows: for each constraint, consider the timepoint specified by the initial solution, say t; then the interval will be $[t-t1, t+t2]$, where $t1$ and $t2$ are random numbers between 0 and 40. Finally, the preference function in each constraint is specified by taking the default one and changing its three parameters a, b, and c, by, respectively, 20%, 25%, and 30%.

To compare our generator with the usual one for classical CSPs, we notice that the maximum expansion (max) for the constraint intervals roughly corresponds to the tightness. However, we do not have the same tightness for all constraints, because we just set an upper bound to the number of values allowed in a constraint. Also, we do not explicitly set the domain of the variables, but we just set the constraints. This is in line with other temporal CSP generators, like the one in (?).

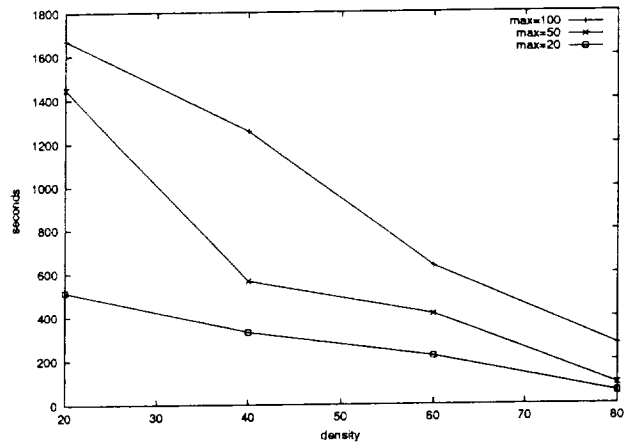

Figure 2: Time needed to find an optimal solution (in seconds), as a function of density (d). The other parameters are: n=50, r=100, pa=20, pb=20, and pc=30. Mean on 3 examples.

In Figure ?? we show some results for finding an optimal solution for STPPs generated by our generator, which has been developed in C++ and tested on a Pentium III 1GHz. As it can be seen, this solver is very slow. The main reason is that it uses a pointwise representation of the constraint intervals and the preference functions. This makes the solver more general, since it can represent any kind of preference functions, even those that don't have an analytical representation via a small set of parameters. In fact, even starting from convex quadratic functions, which need just three parameters, the first solving phase, which applies path-consistency, can yield new preference functions which are not representable via three parameters only. For example, we could get semi-convex functions which are generic step functions, and thus not representable by giving new values to the initial three parameters.

A solving module for STPPs based on a chopping procedure

The second solver for STPPs that we have implemented, and that we will call 'chop-solver', is based on the proof of tractability for STPPs, with semi-convex preference functions and idempotent multiplicative operator of the underlying semiring, described in (?). Let's briefly recall the main argument. The first step is to obtain an STP from a given STPP. In order to do this, we reduce each soft constraint, $\langle I, f\rangle$, of the STPP into a simple temporal constraint. Consider $y \in A$, a value in the set of preferences. Then, since the function f on the soft constraint is semi-conex, the set

Algorithm chop-solver

```
1. input STPP P;
2. input precision;
3. integer n=0;
4. real lb=0, ub=1, y=0;
5. if(CONSISTENCY(P,y))
6.   y=0.5, n=n+1;
7.   while (n<=precision)
8.    if(CONSISTENCY(P,y))
9.      lb=y, y=y+(ub-lb)/2, n=n+1;
10.   else
11.     ub=y, y=y-(ub-lb)/2, n=n+1;
12.  end of while;
13.  return solution;
14. else exit;
```

Figure 3: Algorithm chop-solver

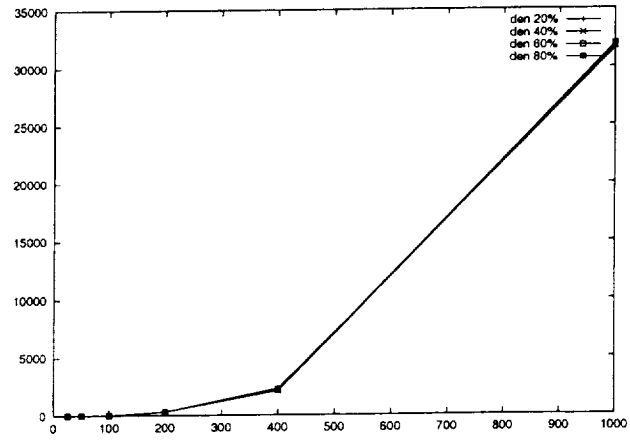

Figure 4: Time, in seconds, (y-axis) required by chop-solver to solve, varying the number of variables (x-axis) and the density, with r=100000 max=50000, pa=5, pb=5 e pc=5. Mean on 10 examples.

$\{x : x \in I, f(x) \geq y\}$ forms an interval, i.e. a simple temporal constraint. Performing this transformation on each soft constraint of the original STPP we get an STP, wich we refer to as STP_y. The proof states that the set of solutions of the STP_{opt}, where *opt* represents the highest level at which the derived STP is consistent, coincides with the set of optimal solutions of the STPP.

The solver we have implemented works with STPPs with semi-convex quadratic functions (lines and convex parabolas) based on the fuzzy semiring. This means that the set of preferences we are considering is the interval [0,1]. The solver finds an optimal solution of the STPP identifying first STP_{opt} and returning its earliest or latest solution. *Opt* is found by performing a binary search in $[0, 1]$. The bound on the precision of a number, that is the maximum number of decimal coded digits, explains why the number of search steps is always finite. Moreover, our implementation allows the user to specify at the beginning of the solving process the number n of digits he wants for the optimal solution's preference level. Figure **??** shows the pseudo-code for this solver.

The search for the optimal preference level starts with $y = 0$. Since STP_0 is the STP we would obtain considering all the soft constraints as hard constraints, that is, with preference function equal to 1 on the elements of the interval and to 0 everywhere else, the algorithm first checks if the hard part of the problem is consistent. If it is found not to be consistent the algorithm stops informing the user that the whole problem is inconsistent. Otherwise the search goes on. Three variables are maintained during the search: ub containing the lowest level at which an inconsistent STP was found, lb containing the highest level at which a consistent STP was found and y for the current level at which we need to perform the "chopping". The three values are updated depending on the outcome of the consistency test.

The actual chopping and the consistency test on the STP obtained are performed by function CONSISTENCY. It receives, as input, the level at which the chop must be performed and the STPP. For each constraint of P it looks at what type is the preference function, a constant, a line or a semi-convex parabola. It then finds the intersection of the function with the constant function at the chopping level. As it finds the intersection for each constraint it fills in the distance matrix F. This matrix is $N \times N$, where N is the number of variables of the problem. It represents the distance graph of the STP (**?**). This means that if the constraint between variable i and variable j is the interval $[a, b]$, then $F[i][j] = b$ and $F[j][i] = -a$. At this point we apply the theorem that states that an STP is consistent if and only if its distance graph has no negative cycles, see (**?**) (**?**) (**?**). In order to accomplish this we run Floyd-Warshall's all-shortest-paths algorithm on F and then check the diagonal elements. If no diagonal elements are negative, we can conclude that STP_y is consistent. If we have already reached the number of decimal digits the user wanted, then we return either the earliest or the latest solution, respectively corresponding to the assignments $x_i = -F[i][0]$ and $x_i = F[0][i]$. If instead one or more diagonal elements are negative, we can conclude that the STP_y is inconsistent and either return the solution of the last consistent STP or keep searching at lower levels of preference. The solution we return is always made of integers, that is, in the case of the earliest solution, the real numbers found intersecting the preference functions with the chopping level are approximated to the first larger integer while for the latest the approximation is to the largest smaller integer.

Figure **??** shows some experimental results for chop-solver. We have used basically the same random generator used to test the solver described in Section 3, although it has been slightly modified since the two solvers use two different representation of a constraint.

We have tested chop-solver by varying the number of variables, from a minimum of 25 up to a maximum of 1000, and

the density from 20% to 80%.

¿From Figure ?? we can conclude that chop-solver is only slightly sensitive to variations in the density and, in this sense, it finds more constrained problems a little more difficult. This fact can be partially explained by the way problems are generated. Having a higher density means having more constraints with non trivial parabolas, i.e. $a \neq 0$. The intersection procedure in this case is a little more complicated than in the case of constant or lines.

Chop-solver is indeed sensitive to the number of variables since it yields an increase of the number of constraints on which the intersection procedure must be performed.

The choice of mantaining a fixed maximum enlargement of the intervals, that can be interpreted as a fixed tightness, is justified by the continuos representation of the constraint this solver uses. In fact, each constraint is represented by only two integers for the left and right ends of the interval and 3 doubles as parameters of the function. Increasing max affects this kind of representation of a constraint only making these values bigger in modulo. This change however does not affect any of the operations performed by chop-solver.

Path-solver vs. chop-solver

In Table ??, ?? and ?? we can see a comparison between chop-solver and

path-solver.

	D=20	D=40	D=60	D=80
path-solver	515.95	235.57	170.18	113.58
chop-solver	0.01	0.01	0.02	0.02

Table 1: Time in seconds, used by path-solver and chop-solver to solve problems with $n = 30$, $r = 100$, $max = 50$, $pa = 10$, $pb = 10$, and $pc = 5$ and varying density D.Results are mean on 3 examples.

	D=20	D=40	D=60	D=80
path-solver	1019.44	516.24	356.71	320.28
chop-solver	0.03	0.03	0.03	0.03

Table 2: Time in seconds, used by path-solver and chop-solver to solve problems with $n = 40$, $r = 100$, $max = 50$, $pa = 10$, $pb = 10$, and $pc = 5$ and varying density D.Results are mean on 3 examples.

	D=20	D=40	D=60	D=80
path-solver	2077.59	1101.43	720.79	569.47
chop-solver	0.05	0.05	0.06	0.07

Table 3: Time in seconds, used by path-solver and chop-solver to solve problems with $n = 50$, $r = 100$, $max = 50$, $pa = 10$, $pb = 10$, and $pc = 5$ and varying density D.Results are mean on 3 examples.

It appears clear that chop-solver is much faster. It is also true that, in a sense, it's also more precise since it can find an optimal solution with a higher precision. It must be kept in mind, though, that path-solver is more general. In fact, the point-to-point representation of the constraints, to be blamed for its poor performance, allows one to use any kind of semi-convex function, e.g. step functions, that cannot be easily compactly parametrized. Keep in mind that such a pointwise representation is required in order to be able to apply path consistency. It is also true that, in general, time is dealt with as a discretized quantity, which means that, once the measuring unit that is most significant for the involved events is fixed, the problem can be automatically cast in the point-to-point representation. Moreover, even wanting to extend the types of parametrized functions in the continuos representation for chop-solver, we must remember that the system deriving from intersecting the constant at chopping level and the function must be solvable in order to find the possible intersections. However, the continuous representation used by chop-solver is, undoubtly, more natural because it reflects the most obvious idea, the idea we all have in mind, of such constraints, that is an interval plus a function over it. The improvement in terms of speed are impressive but the loss in generality is, on the other hand, considerable.

Conclusions and Further work

.....

References

A. Biso, F. Rossi, and A. Sperduti. Experimental Results on Learning Soft Constraints. *Proc. KR 2000*, Morgan Kaufmann, 2000.

S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, March 1997.

R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. Artificial Intelligence, Vol. 49, 1991, pp. 61-95.

S. Haykin. *Neural Networks: a comprehensive Foundation.* IEEE Press, 1994.

L. Khatib, P. Morris, R. Morris, F. Rossi. Temporal Constraint Reasoning With Preferences. *Proc. IJCAI 2001.*

L. Khatib, P. Morris, R. Morris, F. Rossi, A. Sperduti. Learning Preferences on Temporal Constraints: A Preliminary Report. *Proc. TIME 2001*, IEEE Computer Society Press, 2001.

A.K. Mackworth. Constraint satisfaction. In Stuart C. Shapiro, editor, *Encyclopedia of AI (second edition)*, volume 1, pages 285–293. John Wiley & Sons, 1992.

R. Shostak. Deciding linear inequalities by computing loop residues. J. ACM 28 (4) (1981) 769-779.

C.E. Leiserson and J.B. Saxe. A mixed-integer linear programming problem which is efficiently solvable. proceedings 21st Annual Allerton Conference on Communications, Control, and Computing (1983) 204-213.

Y.Z. Liao and C.K. Wang. An algorithm to compact a VLSI compact symbolic layout with mixed constraints. IEEE

Trans. Computer-Aided Design of integrated Circuits and Systems 2 (2) (1983) 62-69.

F. Rossi and A. Sperduti. Learning solution preferences in constraint problems. *Journal of Experimental and Theoretical Computer Science*, 1998. Vol 10.

T. Schiex. Possibilistic constraint satisfaction problems, or "how to handle soft constraints?". In *Proc. 8th Conf. of Uncertainty in AI*, pages 269–275, 1992.

E. Schwalb, R. Dechter. Coping with disjunctions in temporal constraint satisfaction problems. In *Proc. AAAI-93*, 1993.

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.